RTW SUPPORT FOR PARALLEL 64bit ALPHA AXP-BASED PLATFORMS

Christian Vialatte, Jiri Kadlec,

Introduction

Presentation of software supporting the Real-Time Workshop (Matlab 5.3), targeting AD66 ISA and AD66-PCI boards. These boards are using 64-bit Alpha AXP processor, operating on 233MHz. Parallel systems with multiple Alpha boards can be created. Alpha nodes communicate via dedicated Transputer links or C44-based comports. The main emphasis of the paper will be on the targeting multiprocessor platforms from the Simulink and RTW. The presented solution is based on compilation of several SIMULINK diagrams for special, dedicated Alpha targets, connected via special I/O-like communication blocks. Our aim is not to show that DEC designs faster processors than Intel, but to demonstrate that impressive speed-up can be achieved by parallelising and by using operating system free processors.

Using Simulink and RTW in Real-Time

Simulink is an excellent tool for developing algorithms and simulations but lacks speed and most importantly the ability to run in real-time. Conscious of the growing need for rapid prototyping, the Mathworks developed RTW to quickly turn Simulink diagrams into C code. This is definitely the basis for producing effective applications but RTW does not provide solutions to shortcomings inherent to the host operating system. Most operating systems (including Windows 9x, NT and Linux) do not offer any guarantees for real-time latency. Due to their multi-tasking ability, operating systems must share the CPU time between all running processes and decide alone when and for how long each process is executed. Typically, an application will run every few milliseconds but if the OS feels like doing something else this will create some unexpected delay.

Windows, for instance is not designed as a real-time operating systems and cannot typically run real-time applications satisfactorily. However, it is possible to program it at low level (ring 0) by-passing the operating system scheduler. The task is not easy because none of the usual tools and OS services taken for granted (source debugger, disk and screen access,…) are not anymore available at ring 0. More importantly a mistake will generally end up crashing the system making any development a arduous job. Windows allows programmers to virtualise interrupts and execute code which cannot be descheduled by the OS and not even interrupted by some software or hardware. This is the mechanism used by Humusoft in Windows Target. By writing a ‘user extensible’ device driver, Humusoft brought the ability to run a RTW model in real-time to Windows. Windows Target turns the PC into an hybrid system partially running in real-time.

We are introducing another approach based on the use of a multi-processor system devoid of the above mentioned limitation.

System Description

A typical system is made of any number of ADxxx boards. Each board is a building block for parallel network of alpha processors and is made of a processing engine (alpha processor ranging from 233 MHz to 667 MHz) with cache and main memory, and a link engine dedicated to inter-processor communications (Transputer, C44 DSP processors or fibre channel links). The first board of the system (root board) is talking to the PC host via a PCI interface. All the boards are inter-connected via dedicated point to point physical links.

Each board in the system runs a minimal real-time kernel offering the full processing power of the processor to the user application. The kernel guaranties that no unrequested OS-like background tasks are taking place. The user application fully owns the system resources (memory, CPU, PCI bus). So, this system is suitable for real-time and response time will not be harmed by unpredictable background jobs.

ADxxx board:
The ADxxx boards used by the Parallel RTW software have been designed for compute-intensive applications, such as data mining, DSP or image processing. The DEC 21066 and (21164) Alpha processor is a fully pipelined, dual-issue (quad-issue) 64-bit advanced RISC processor. Running at 233MHz (667 MHz), it is capable of a maximum 233 Mflops (1334 Mflops). It includes an integral memory management unit and cache controller, IEEE and VAX-compatible floating-point and a PCI I/O controller. Root boards include an ISA or PCI interface to the host PC. See the User Guide [2] [3] for more details about the board.

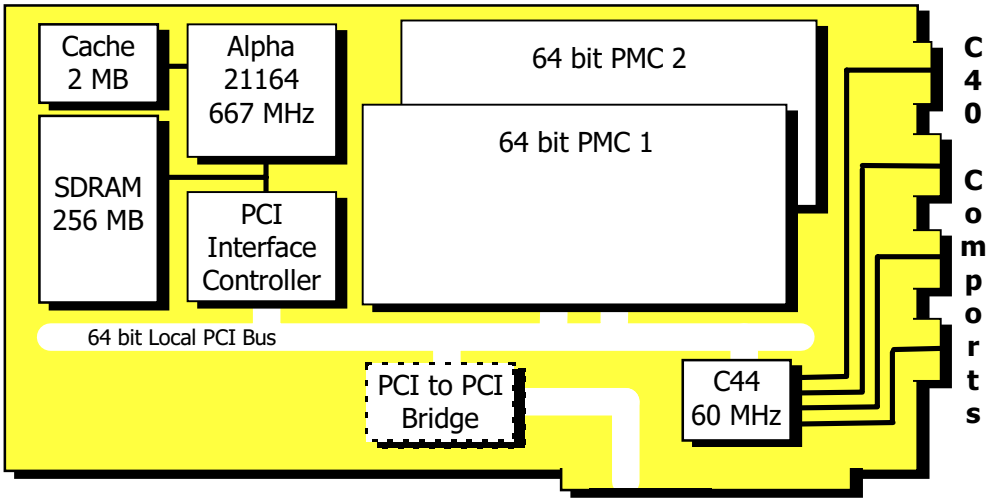

Figure 1: AD164 Block Diagram

3L C Compilers and Software:
The key software development tool is the 3L Parallel C [1], and the DEC ANSI C compiler. The DEC compiler runs in the Alpha processor, supported by a simple DOS, Windows 9x, NT or Linux server. A minimal subset of the DEC OSF is emulated for the compiler. Therefore, the sequential core of compiler is the OSF optimising ANSI C from Digital Equipment. In addition, the standard 3L Parallel C functions known from the Transputer and C40 world can be used.

Parallel-RTW

We propose to use the above hardware system to bring scalable performance to Simulink/RTW. The written software is able to manage communications between several models each targeted to one processor. Efforts have been made to keep the user-friendliness of RTW and the generation of alpha parallel code is straight forward. The Parallel-RTW software provide automatic support for all the code generated by RTW, a single button click from the build menu in Simulink will automatically generate an alpha application.

It would be an extremely difficult task to try and devise a generic way to split the C code generated by RTW to run on several processors. A much easier approach is to do the splitting at high level, in Simulink, by using inter-model exchange blocks. Three specific Simulink blocks have been designed for that purpose:

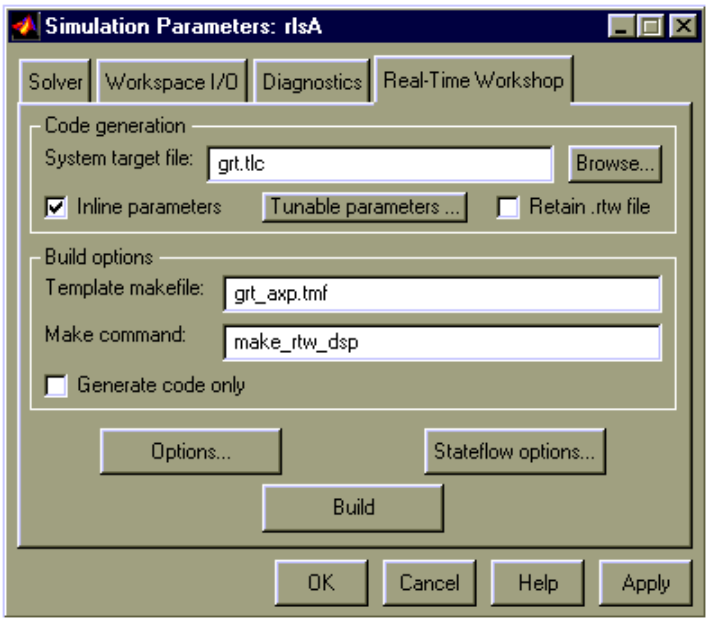

Figure 2: Compiling Option Dialogue Box

- 'Init' block: an initialisation block doing the proper declaration in the model.c code generated by RTW
- 'ChanMaster' block: block initiating a data exchange with another model over a logical/physical channel. The channel will automatically be logical if the two models are targeted to the same ADxxx board, and physical if two boards are involved. The block first send its input data to its corresponding 'ChanSlave' block in the receiving model, then receive the answers resulting from the computation made on the previous set of data. Note that a side effect of using a 'ChanMaster' block is to insert a Unit Delay (1/z) on each of its input line.
- 'ChanSlave' block: receiving end of a 'ChanMaster'. Firstly, new inputs are received then the results computed from the previous inputs are sent back to the 'ChanMaster' block. The number of the channel, the width of inputs and outputs are defined by the user as parameters for the 'ChanMaster' and 'ChanSlave' blocks. Of course these parameters must match if the data exchange is to function correctly: 'ChanMaster' must send the number of information expected by the 'ChanSlave' waiting on the same channel, and the other way round. The slave model StopTime is forced to 0 (infinite) and its sample time is imposed by the master frequency in sending data.

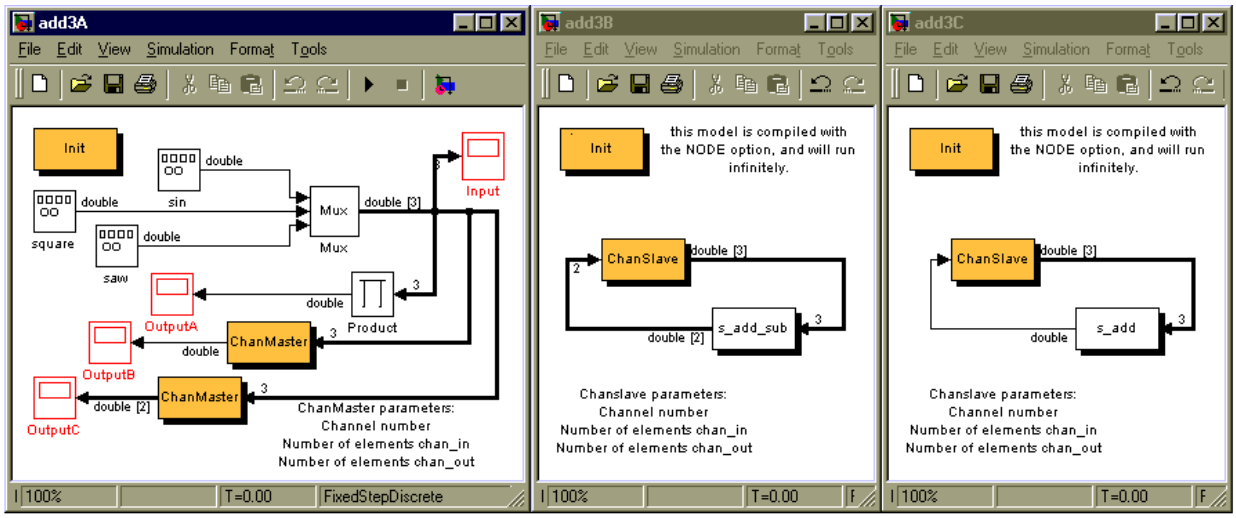

Figure 3: Three Processor Parallel Models

Let's imagine we want to execute three operations on the output of some signal generators and display the results in three oscilloscopes. Two 'ChanMaster' blocks send the 3 generator signals to two slave models, and receive respectively 2 and 1 signals computed by the slaves. The slave models get the input signals, perform some computations and return respectively 2 and 1 signals. The computation can be any combination of Simulink blocks

or C coded S-functions. In this case the S-function have been coded using the TLC code generation tool [4]. This example (shown in figure 3) is only designed to show the parallelising mechanism and the fact that a 'ChanMaster' block can be seen as a sub-system hiding the corresponding model.

Performance

To test the performance we have chosen the standard RLS noise cancellation demo provided as a DSP blockset demo with Matlab. An input signal is mixed with some random noise and a RLS block is trying to restore it (Figure 4).

We have split the model into two and placed each part on a processor. The cutting is very simple, on one side the signal generators, noise filter, scope and on the other the RLS block (Figure 5). To maintain the equivalence between the original and parallel version of the system, a Unit Delay block (1/z) must be inserted on each of the 'ChanMaster' block input line.

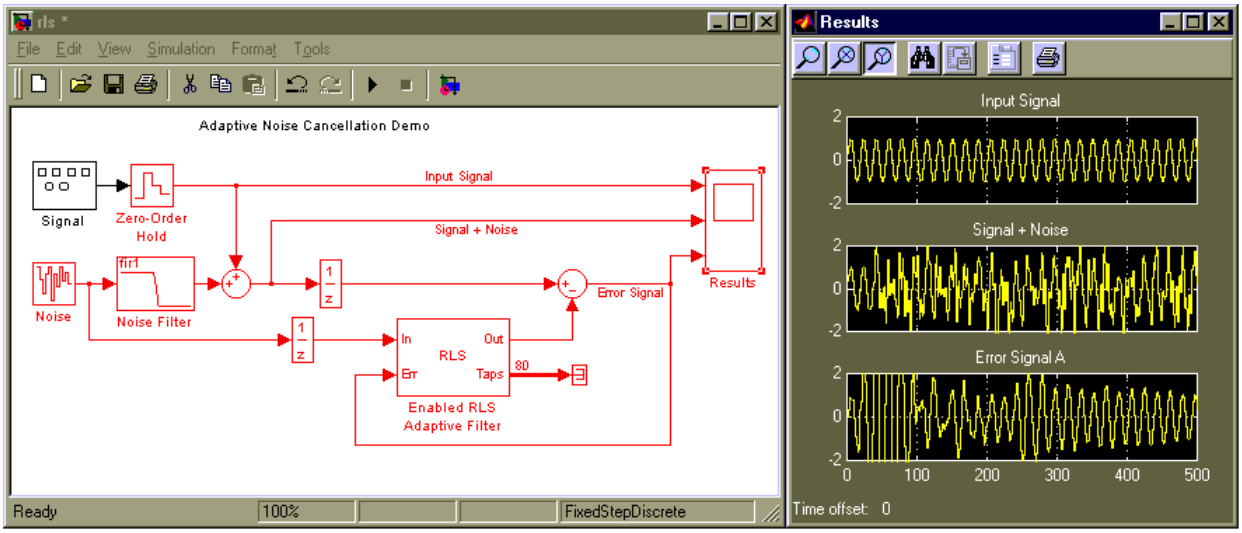

Figure 4: Original RLS Demo (with 1/z blocks)

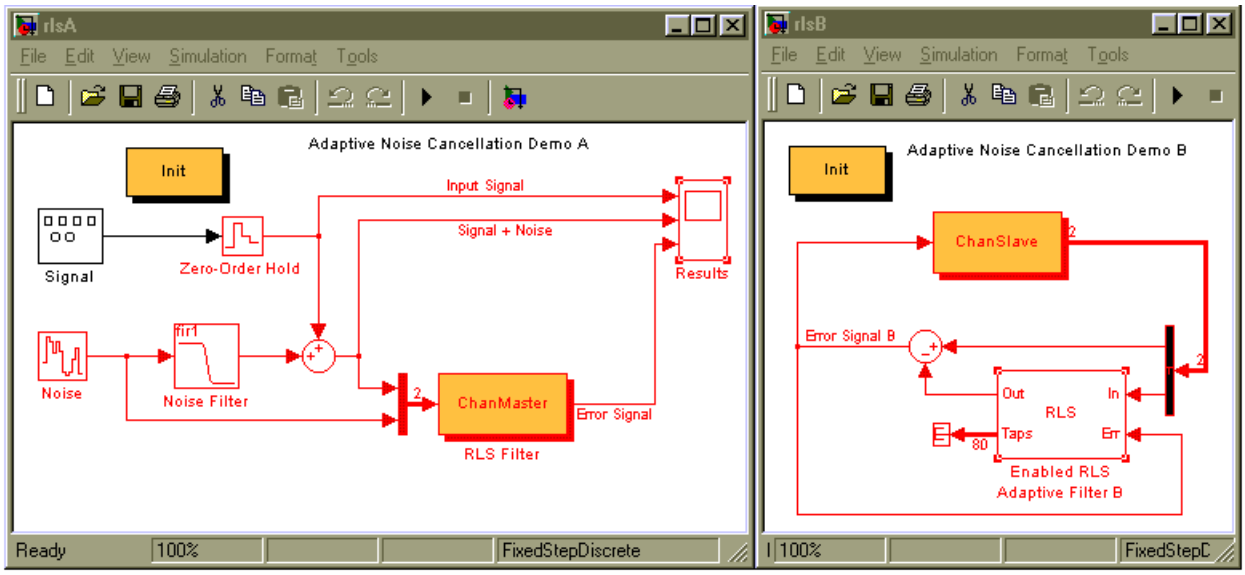

Figure 5: Parallel Version of the Original RLS Demo

In the above case the work load happen to be far too unbalanced so, in order to show the speed-up in a case of ideal distribution we have extended the model with a second RLS sub-system. Several RLS filters can be used in combination with a selection algorithm deciding which adaptive method is the most appropriate (figure 6). The figure 7 shows the same system split into two models.

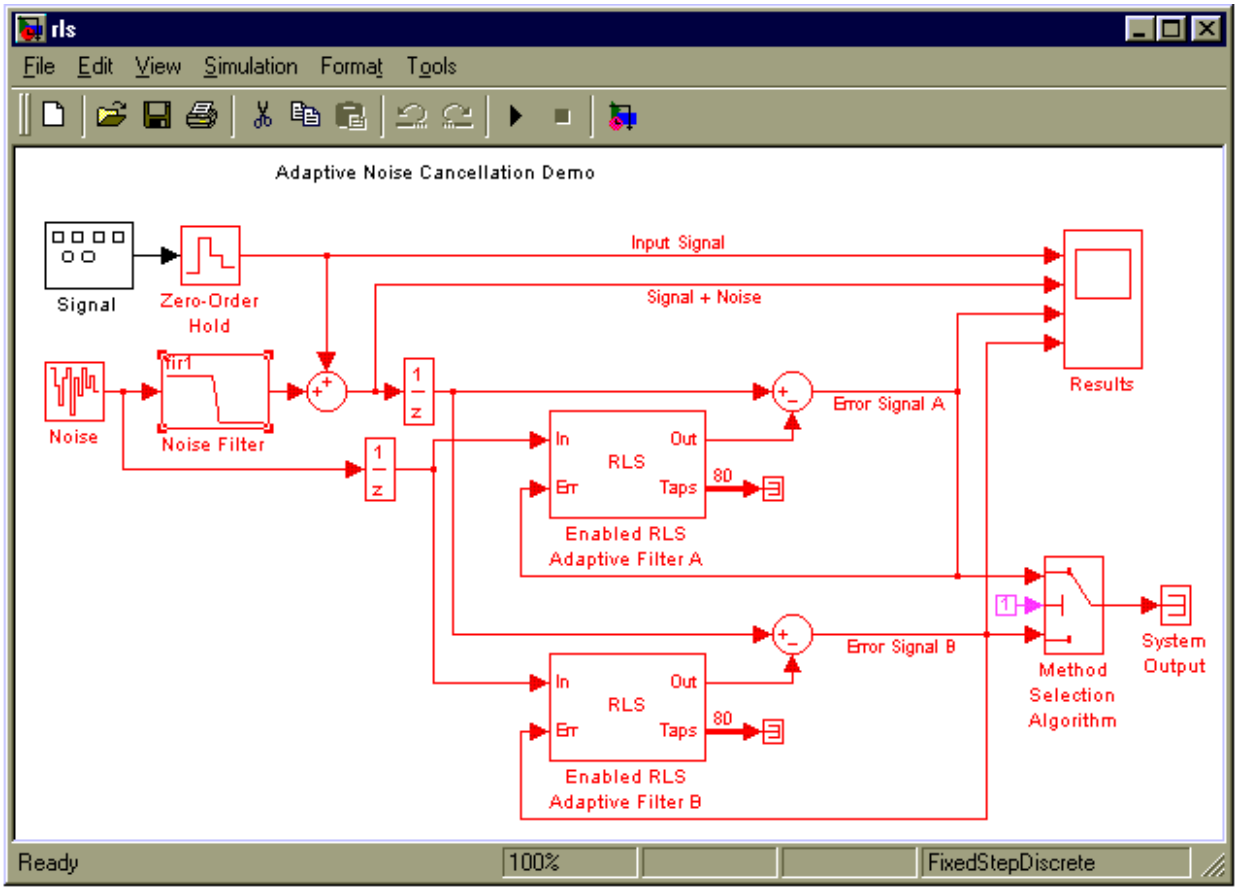

Figure 6: Extended RLS Model

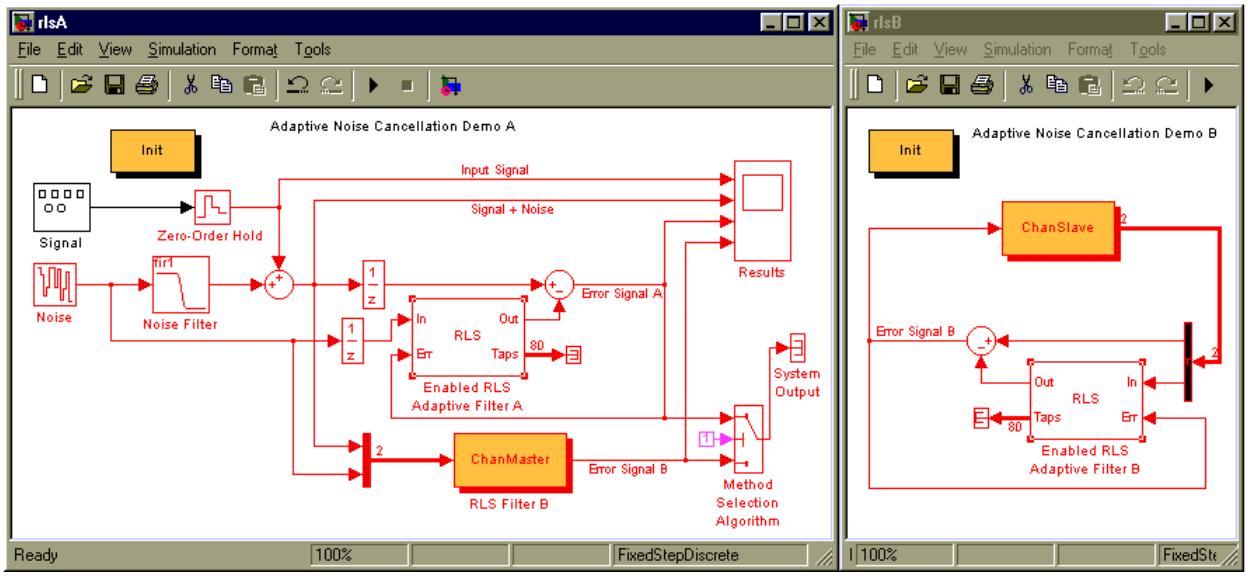

Figure 7: Parallel Version of the Extended RLS Model

To run the different test models we used a 333MHz Pentium 2 PC fitted with one ad66PCI root and an AD66 node both running at 233MHz. The goal is not to show the performance of a single board system but the speed-up achieved by scaling up the system. The models have been compiled by RTW using the grt.tlc generic system target file.

By varying the FIR length of the RLS block, we can modify the load of work for each model. The table below shows the timings in micro seconds for different values:

	Original Model with delays		**Split Models on 1 processor**		**Split Models on 2 processors**	
	Time	**Ratio**	**Time**	**Ratio**	**Time**	**Ratio**
1 RLS block, FIR length 30	618	1	762	1.23	647	**1.05**
1 RLS block, FIR length 80	4463	1	4834	1.08	4274	**0.96**
2 RLS block, FIR lengths 30/30	1261	1	1354	1.07	757	**0.60**
2 RLS block, FIR lengths 30/80	5558	1	5426	0.97	4277	**0.77**
2 RLS block, FIR lengths 80/30	5172	1	5401	1.04	4818	**0.93**
2 RLS block, FIR lengths 80/80	9762	1	9883	1.01	4818	**0.49**

1 RLS block:
We can see that the work load for the 1 RLS system is unbalanced and that the second processor has a much bigger task. Even with a heavy load (FIR length 80), the gain achieved by parallelism hardly compensate for the cost of splitting the model which introduces communication overhead. A closer to optimum cut can be achieved by splitting the RLS filter itself, this involves some mathematical work but can be done. This is an extreme case where the master model has nearly nothing to do and spends its time waiting for the slave model to be ready to transmit its results.

2 RLS blocks:
The table shows an impressive speed-up for the two balanced settings (FIR length 30/30 and 80/80). Then we see that if the load of work is light, the communication overhead becomes significant and the parallelism proves less effective but still remarkable. The two other settings (30/80 and 80/30) make the system unbalanced and the speed-up suffer from it. Surprisingly enough the 80/80 model speed-up ratio of 0.49 is higher than the maximum expected of 0.50. We can also observe that the original 1RLS/80 model doesn't compare to the original 2RLS mode 80/80 with a ratio of while the split models do. These can be due to the way RTW generates the code (shouldn't the one model 30/80 produce the same timing as 80/30?).

Conclusion:

The most important criteria is the balance of load of work among the different models. If this is satisfied, impressive speed-up can be achieved. However, due to the overhead incurred by the older link technology, the present solution is more adapted to models running heavily computational blocks at low frequency (no more than 1KHz). A similar system built with AD164 boards, the AD66's successor, would give better results at all levels (faster computation, faster links technology) and would be able to run the same binary code.

The next step will be to integrate this mechanism to Windows Target resulting in several benefits:

- access to a full range of i/o cards already supported by Windows Target
- exploit the power of the PC which is currently only used as a terminal
- provide Windows Target with the ability to run massively computational DSP blocks

Contact

Department of Adaptive Systems, Institute of Theory and Automation
Academy of Sciences of the Czech Republic (UTIA AV CR)
Pod vodarenskou vezi 4, 182 08, Prague 8, Czech Republic
email: cv@utia.cas.cz, kadlec@utia.cas.cz

Grants

This work was partially supported by the EU Esprit LTR project No. 33544 ”HSLA” and grants of the Ministry of Education of the Czech Republic OK314-99 and OK317-99.

References

[1] Parallel C/AXP V1.1.1. Parallel C, User Guide, Digital Equipment Corporation Alpha/AXP, 3L Ltd., Edinburgh. UK, 1995.
[2] AD-66 Parallel Processor Boards. User Guide, ALPHA DATA, parallel systems ltd, 58 Timber Bush Edinburgh, EH6 6QH, Scotland, 1995.
[3] AD164 Parallel Processor Boards. User Guide, ALPHA DATA, parallel systems ltd, 58 Timber Bush Edinburgh, EH6 6QH, Scotland, 1998.
[4] Target Language Compiler, Reference Manual, The MathWorks, Inc. Natick, Mass 01760, 1999.